12-2017

Query Expansion Techniques for Enterprise Search

Eric M. Domke
Grand Valley State University

Query Expansion Techniques for Enterprise Search

Eric M. Domke

A Thesis Submitted to the Graduate Faculty of

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

Master of Science in Computer Information Systems

Padnos College of Engineering and Computing

December 2017

Abstract

Although web search remains an active research area, interest in enterprise search has waned. This is despite the fact that the market for enterprise search applications is expected to triple within the next six years, and that knowledge workers spend an average of 1.6 to 2.5 hours each day searching for information. To improve search relevancy, and hence reduce this time, an enterprise-focused application must be able to handle the unique queries and constraints of the enterprise environment. The goal of this thesis research was to develop, implement, and study query expansion techniques that are most effective at improving relevancy in enterprise search.

The case-study instrument used in this investigation was a custom Apache Solr-based search application deployed at a local medium-sized manufacturing company. It was hypothesized that techniques specifically tailored to the enterprise search environment would prove most effective. Query expansion techniques leveraging entity recognition, alphanumeric term identification, intent classification, collection enrichment, and word vectors were implemented and studied using real enterprise data. They were evaluated against a test set of queries developed using relevance survey results from multiple users, using standard relevancy metrics such as normalized discounted cumulative gain (nDCG). Comprehensive analysis revealed that the current implementation of the collection enrichment and word vector query expansion modules did not demonstrate meaningful improvements over the baseline methods. However, the entity recognition, alphanumeric term identification, and query intent classification modules produced meaningful and statistically significant improvements in relevancy, allowing us to accept the hypothesis.

Contents

List of Tables

List of Figures

1 Introduction

Companies generate and maintain copious amounts of digital data. This data takes a multitude of forms including personnel records, purchase orders, design documentation, sales reports, marketing materials, employee handbooks, and so much more. Some of this data is structured and housed in one or more databases, while even more data is kept in unstructured files such as Word, Excel, and PowerPoint documents. With each passing year, it gets easier and cheaper to generate even more data. This makes it more difficult for the average employee to find the information he/she needs to perform his/her job. Over the years, it has been estimated that the average worker spends 1.6 to 2.5 hours each day searching for the information they need [20]. Companies that make the right information readily available can react to business opportunities, on-board staff, and acquire other companies more quickly than their competitors.

When companies need to aggregate and interpret numeric data, they reach for analytics and business intelligence platforms that warehouse structured data in a format that facilitates ad-hoc analysis. However, employees often need to browse through textual information, find a specific document, or get an answer to a specific question. To solve these problems, enterprise search applications are required. While individual business systems often contain search components for finding information within that system, this thesis will focus on enterprise search applications that help employees find information stored in any of the multiple databases and document repositories found throughout an enterprise.

1.1 Challenges in Enterprise Search

Currently, interest in the field of enterprise search is waning. One of the primary information retrieval conferences, the Text REtrieval Conference (TREC), has not hosted a track devoted to enterprise search since 2008 [15]. Google is sunsetting support for its Google Search Appliance, an enterprise search offering meant to be installed on-premise [13]. In

addition, long-time enterprise search blogger Stephen Arnold is planning to wind down his coverage of the industry in light of perceived stagnation in the field [2]. The lack of interest in enterprise search despite continuing research in the web search domain could be due to persistent challenges in enterprise search including:

- *Permissions*: While web search engines (such as Google and Bing) index publicly-accessible information, many critical enterprise documents are only accessible to a subset of employees. In fact, every document found in an enterprise has access controls which define who can view the document. This means that enterprise search applications must restrict results based on what a given user has access to view. Ideally, a search application must also restrict information returned as categories, result counts, or other metrics, so the user cannot infer information about results he/she cannot access.

- *Poor data quality*: Due to the economic incentives of getting your web site to the top of Google's search results, an entire industry has developed around generating high-quality, easy-to-index content on the public web. However, those same incentives rarely exist in the enterprise. Users tend to focus on performing their jobs today without much concern regarding how other employees will be able to find and consume their documents tomorrow. Even when users strive to generate quality content, there is often a lack of training, standards, and policies put in place by leadership to drive consistency.

- *Documents are not linked*: Much of Google's famous PageRank algorithm is driven by analyzing the number and quality of links between documents. However, links rarely exist in enterprise documents, and the quality of a particular document is much harder to assess.

Despite waning interest, the market for enterprise search applications is growing. In 2012, the market was valued at $1.5 billion dollars while Frost & Sullivan project that it

will be worth $4.7 billion dollars by 2019 [10]. Research in the field is still needed and relevant to support this growing demand.

1.2 Characterizing Enterprise Search

In order to better characterize the enterprise search problem, query logs were obtained for two existing enterprise search applications at a local medium-sized manufacturing company. One set of data came from a content management system, while the other came from a custom enterprise search solution. This data was compared against query logs from publicly-accessible web search engines and domain-specific databases. The results of the analysis are presented in Table 1. The data shows that different types of queries are more prominent in some search engines than others. For example, URL searches[1] are more common on publicly-facing search applications than those inside the enterprise. The medical search engine PubMed had a higher percentage of advanced searches[2]. Alphanumeric searches[3] (e.g. for part numbers, employee IDs, telephone numbers, etc.) were seen more frequently in enterprise search logs than the logs of other search engines.

A deeper analysis of the query logs for the "Internal L" search engine is shown in Figures 1 and 2. These figures indicate that nearly 90% of the total searches (covering 73% of the distinct searches) for this custom enterprise search application were for some form of person data. Of these person searches measured by volume, most consisted of a full name or last name. However, the distinct search data showed a large percentage of person searches which were either ambiguous or otherwise uncategorized. Alphanumeric searches also played a large role constituting 26% of the distinct searches and 13% of the total search volume. Many of these alphanumeric searches were looking up an employee by ID

[1]URL searches were identified in the query logs by looking for queries that started with "www" or ended with a common URL suffix (e.g. "com", "net", "edu", etc.). As with the other methods of query type identification, it is very possible that this simplistic approach labeled some queries as being URL queries which weren't and vice versa.

[2]Advanced queries are those which contain field constraints – e.g. that the `title` should contain "prehospital" while the `publication date` must be between 2005/09/07 and 2005/10/05

[3]In this analysis, any query where 15% or more of the characters were digits was labeled as alphanumeric

Table 1: Query log statistics for various search engines. Percentages represent estimates of distinct queries belonging to that category.

Data Set	Year	Distinct Queries	Avg. Word Count	Alphanumeric	URL	Advanced
Internal L[a]	2017	61,882	1.5	26%	0%	0%
Internal T[a]	2016	1,117	1.8	31%	0%	0%
AOL[b]	2006	373,903	2.9		17%	
PubMed[c]	2005	724,578	3.5	17%	0.03%	11%

[a] Proprietary data sets from the manufacturer
[b] Obtained from `http://octopus.inf.utfsm.cl/~juan/datasets/`
[c] Obtained from `ftp://ftp.ncbi.nlm.nih.gov/pub/wilbur/DAYSLOG` as documented by Mosa and Yoo, `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3552776/`

or trying to find a part number.

What is common between all search engines is that the searches performed are consistently quite short. Across the different search engines, a given query only contained 2 to 3 words on average with the two enterprise search applications showing the lowest average word counts. This data is consistent with analyses performed on other search applications. Wang and Wang noted that in June of 2016, 1 and 2 word queries accounted for 63% of the search traffic on Bing and 35% of the distinct searches [46]. With so few words, search engines often struggle identifying and correctly ranking relevant documents. As a result, many search engines will enrich queries with additional words, constraints, and boosts in order to improve relevancy in a process known as query expansion.

1.3 Research Question

In the face of poor data quality, unlinked documents, and high user expectations, improving the performance of enterprise search applications is crucial to their success. Since query expansion is a fundamental technique for improving the performance of search engines, the goal of this thesis is to develop, implement, and study query expansion methods, determining which are most effective at improving relevancy in enterprise search. For the purposes of this thesis, *query expansion* will be interpreted broadly to include the addition of any

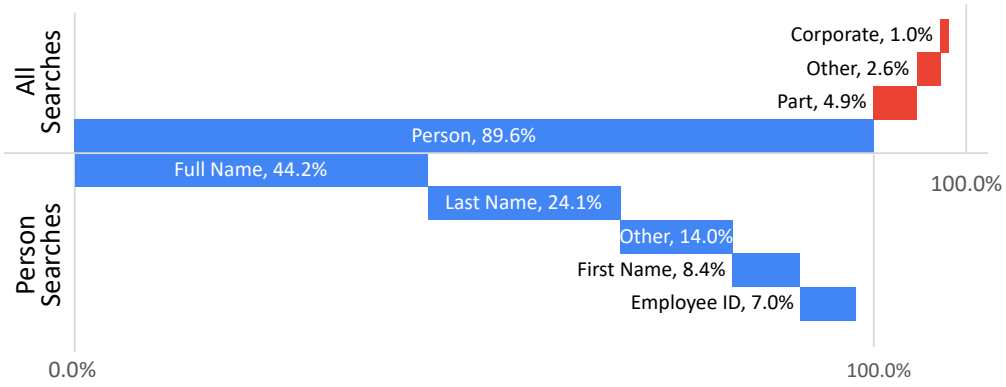

Figure 1: Categorization of 340,000 total enterprise searches for "Internal L" between May and October 2017

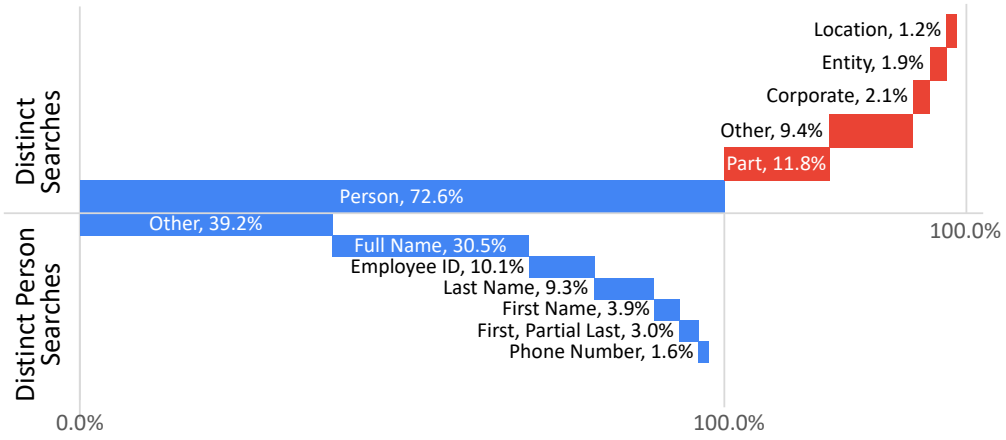

Figure 2: Categorization of 61,881 distinct enterprise searches for "Internal L" between May and October 2017

word(s), conditions, or operators to the user's query with the goal of improving relevancy. These additions might expand, restrict, or merely reorder the result set. It is hypothesized that because enterprise search queries are fundamentally different than the queries submitted to other search applications, query expansion techniques tailored for the enterprise are most effective at improving relevancy over baseline methods. In order to study this hypothesis, a custom enterprise search application deployed at a medium-sized manufacturing company will be studied. This application was built using Apache Solr[4] version 6.5.0.

[4] `http://lucene.apache.org/solr/`

2 Background

To understand the problem of query expansion, it is important to understand how search engines fundamentally operate. At the core of each enterprise search application is a NoSQL database. This database contains collections of largely de-normalized documents, each containing multiple fields. Due to the size and nature of these databases, documents within a collection cannot be queried by any arbitrary field. Rather, documents can only be retrieved using pre-built indices. These "inverted" indices each contain a dictionary of terms with each term pointing to a list of documents containing that term in a particular field. With some metadata fields, the terms in the index represent the entire contents of that field after only minimal processing (e.g. converting the value to lower case). For full-text fields, the terms in the index are individual words contained within the field. These "words" are often stemmed before indexing so that similar word forms (e.g. *quick* and *quickly*) are stored as a single term within the index. To satisfy a user's query, the search application first uses the indices to identify the documents which satisfy the Boolean constraints of the query (e.g. `field1` *should contain* `x` `AND` `field2` *must not contain* `y`). However, once identified, the applicable documents must be ranked so that the user is presented with the most relevant documents first.

2.1 Ranking Algorithms: tf-idf and BM25

One way to approach the ranking problem is to model documents and queries as vectors. Such vector space models utilize V-dimensional vectors where V is the number of words in the vocabulary of the document collection. At each position within the vector, a weight is assigned indicating how much the document or query has to do with that particular word. In the simplest model, term frequencies (tf) are used. However, this overly-simple model assumes that all words are equally important when describing a particular document or query. In reality, common words (e.g. *why*) which occur in many documents are

much less important than topical words (e.g. *engineering*) which only occur in a handful of documents. As a result, the term frequency values are often scaled by inverse document frequency (*idf*) values for each word. Once the vectors are constructed, the similarity between two documents or between a document and a query can be computed by calculating the dot product between the vectors. Since computing an exact dot product can be computationally expensive, a practical formula employed by Apache Lucene (the database at the core of Apache Solr) can be approximated as follows

$$\text{score}(q, d) \propto \sum_{t \in q} (\text{tf}(t, d) \cdot \text{idf}(t) \cdot \text{lengthNorm}(t, d)) \tag{1}$$

which gives the score of document d relative to a query q consisting of terms t [43]. It is observed that a document with twice the number of occurrences of a given word is generally not twice as relevant, so Lucene computes the *tf* score as

$$\text{tf}(t, d) = \sqrt{\text{frequency}} \tag{2}$$

Similarly, inverse document scores do not scale linearly, and one must account for the case where a term does not occur in any documents. Therefore, the *idf* score becomes

$$\text{idf}(t, d) = 1 + \log\left(\frac{\text{docCount} + 1}{\text{docFreq} + 1}\right) \tag{3}$$

Finally, the *lengthNorm* is used so that short documents have scores comparable to long documents.

In addition to using vector models, researchers have also used probability theory to develop ranking algorithms. That is, they have worked to develop formulas which calculate the probability that a particular document is relevant for a given query. One of the most successful such formulas to-date (as documented in experiments by Spärck Jones et al. [41] and others) is the BM25 formula introduced by Robertson, et al. [35]. It derives from the

td-idf formula according to the scheme

$$\text{score}(q,d) = \sum_{t \in q} \left(\text{idf}(t) \frac{\text{tf}(t,d)(k+1)}{\text{tf}(t,d) + k\left(1 - b + b\frac{|d|}{\text{avgdl}}\right)} \right) \tag{4}$$

where *idf* is the inverse document frequency of the query term, *tf* is the frequency of the term within a given document, $|d|$ is the length of the document, *avgdl* is the average document length within the collection, and k and b are constants generally set to 1.2 and 0.75 respectively. Because of its success, BM25 has become the default ranking formula used by Apache Lucene. Its full implementation is described by Pérez-Iglesias [32].

2.2 Naïve Bayes

The document ranking task shares similarities with the document classification task which aims to identify the most likely classification for a given document. The most common machine learning technique used for classification is Naïve Bayes. Naïve Bayes classifiers are built using a probability axiom of the same name which states that the probability of a class, c, given evidence (such as a document or query), d, is given by

$$p(c|d) = \frac{p(d|c)p(c)}{p(d)} \tag{5}$$

Assuming a simple "bag of words" model where documents consist of words with independent probabilities, this equation becomes

$$p(c|d) = \frac{d_c}{docCount} \prod_{t \in d} \frac{\text{tf}(t,c)}{V_c} \tag{6}$$

where d_c is the number of documents in class c, *docCount* is the total number of documents, $\text{tf}(t,c)$ is the number of times term t occurs in class c, and V_c is the total number of terms in class c. This formulation shares the notion of term frequency with the BM25 ranking algorithm in Equation 4. Such similarities are not coincidental as Manning notes

that the binary independence model (a predecessor to BM25) is equivalent to a multivariate Bernoulli Naïve Bayes model [28]. In addition, while the traditional formulation of Naïve Bayes does not consider factors such as document length normalization or inverse document frequency, which are included in BM25, Rennie et al. has found that including these factors into the Naïve Bayes model does improve its performance [34]. Therefore, there is reason to believe that Naïve Bayes and BM25 models can be used interchangeably in certain circumstances.

2.3 Query Expansion

Query expansion is an old research topic with the earliest papers dating as far back as the 1960s. The primary goal of query expansion is to increase the percentage of relevant documents which are returned by the query (i.e. increase recall). Due to its long history, query expansion has been discussed in depth by many authors including Manning et al. [28], Soni and Singh [40], and Ooi et al. [29]. While many variations exist, most query expansion techniques follow one of two approaches – relevance feedback and knowledge structures (thesauri and ontologies).

The relevance feedback technique was first popularized by Rocchi in 1971 [36]. It involves using information about known relevant documents to adjust the query vector so that it is more similar to the vectors of known relevant documents than those of known non-relevant documents. (Alternatively, Manning notes that a probabilistic formulation using Naïve Bayes could also be used [28].) This approach assumes that the relevant documents tightly cluster around a single topic which is distinct from the non-relevant documents. To identify the relevant documents, either the user is prompted to give relevancy judgments interactively (explicit feedback), documents frequently selected in the search results are assumed to be relevant (implicit feedback), or the top-k results are assumed to be relevant (pseudo feedback). Because users rarely want to be burdened with providing additional information, one of the latter two approaches is normally used. Much research has

taken place regarding how to improve these techniques. Hahm has explored using implicit feedback to develop an ontology-based user profile for aiding query expansion and document retrieval [18]. With regards to pseudo relevance feedback, Singh has investigated using a support vector machine (SVM) classifier to improve the top-k results [38], Song has examined the usage of key phrases as opposed to isolated terms [39], and Ye investigated new quality-level metrics for identifying the most useful documents [49].

2.4 Thesauri, Ontologies, and Word Vectors

While relevance feedback identifies related terms after the query has been executed, knowledge structures allow expansion before query execution. Thesauri are knowledge structures which encode flat word association rules (e.g. *TV* is a replacement for *television* or *boss* is synonymous with *supervisor*). These structures can either be built manually through research into a particular domain, obtained from an external source, or generated through automated techniques where they are learned from a corpus or search logs. One approach to automatic thesaurus generation described by Manning et al. involves using a term-document matrix to determine the similarity score between two terms [28]. The dimensionality of this matrix can be reduced via latent semantic analysis techniques. This approach is used by Stanford's GloVe framework[5]. Another approach involves learning a vector representation for words in a corpus using a neural network trained on the words found within k words of the target word. This approach is used in several programs such as Google's word2vec[6] and Facebook's fastText[7]. Regardless of the approach, Manning et al. found that automatically-generated thesauri tend to have issues with the quality of associations leading to high false positive and false negative associations. Just the same, Cui et al. found success identifying synonyms using term co-occurrence in query logs for general web searches [12]. On the narrower domain of U.S. patent searches, Tannenbaum and

[5] `https://nlp.stanford.edu/projects/glove/`
[6] `https://code.google.com/archive/p/word2vec/`
[7] `https://github.com/facebookresearch/fastText`

Rauber also achieved success mining term associations from query logs [42].

Relative to thesauri, ontologies describe richer, graphical structures which relate broader terms with narrower ones. Just like thesauri, ontologies can be built manually by domain experts, extracted from query logs, or obtained from an external source, such as WordNet[8]. Because WordNet is a freely available general ontology of the English language, it is often studied by researches. Both Gong et al. [16] and Audeh [3] have been able to use it successfully in their query expansion research. However, because WordNet often contains multiple (sometimes contradictory) senses of the same word, using it indiscriminately can hurt performance as Gong noted. Ontologies can be used for more than just suggesting additional query terms. In particular, Andreou used the disambiguation capabilities of an ontology to re-rank expansion terms instead of suggesting new terms [1].

2.5 Neural Networks

Artificial neural networks (like the ones used to learn word vectors) form a class of machine learning algorithms based on a simplified model of the human nervous system. In particular, a network is designed consisting of multiple layers of "neurons" with at least one input layer, a hidden layer, and an output layer. Each neuron attempts to learn the parameters of an activation function for determining whether to "trigger" connected neurons. Prior to training, each neuron is initialized with a random weight for its activation function. During training, a numeric vector representing the features of each training example is presented to the input layer, and the outputs are computed based on application of the activation functions. The output vector generally represents the predicted classification of the input with a single neuron responsible for reporting the likelihood of each class. The difference between the predicted and actual output vectors is then back propagated through the network to adjust the weights in favor of producing the correct output. This process is repeated iteratively so long as the error in the network (the overall differ-

[8] `https://wordnet.princeton.edu/`

ence between predicted and actual outputs) is above a defined threshold and continues to decrease. At some point (e.g. based on the reported error or the performance of the network on held out validation data), training is stopped and the overall performance of the network is evaluated.

2.6 Entity Recognition in Queries

The traditional query expansion techniques outlined in Section 2.3 aim to identify related terms without requiring an understanding of what the terms represent. Many queries contain terms which represent entities. For example, *harry potter* in the query *harry potter walkthrough* represents a "game" while *mountain view* in *hotel in mountain view with pool* represents a "location". Correctly identifying these entities provides additional avenues for query expansion. In the enterprise search domain, Liu et al. researched using conditional random fields (CRF) to extract entities from unstructured data, combining evidence from structured and un-structured sources to build a graph model of entities and their relationships. From this model, a ranked list of entities related to those identified in the query were suggested as expansion terms [27]. In the web search domain, Brandão achieved success retrieving Wikipedia articles for entities found within a query and using terms from the info boxes as query expansion terms [6].

To exploit these expansion techniques, one must first identify the entities present in the query. This problem can be approached in a couple of different ways. If the "entities" are actually field values from the relevant documents, then the problem can be considered one of mapping values to the correct fields. For this problem, Kim et al. used a Naïve Bayes classifier (discussed in Section 2.2) to identify the most likely fields for movie searches across an XML version of the IMDB database [22]. One could also see this as a problem of trying to recognize named entities (i.e. people, locations, organizations, etc.) within the query. In this task, search indices using document ranking algorithms (shown to be similar to Naïve Bayes in Section 2.2) have proven useful. In a later study, Kim and Croft pro-

posed a probabilistic model for identifying relevant fields by first identifying the k most relevant documents using standard search relevancy models and then identifying the fields within those documents which contained specific search terms [21]. Similarly, both Laclavík et al. [24] and Cornolti et al. [9] leveraged search indexes in the named-entity recognizers they submitted for the 2014 ERD challenge. Rüd et al. took a slightly different approach, more analogous to pseudo-relevance feedback. In particular, they fed the snippets returned with the query results into a traditional named-entity recognizer in order to properly identify entities within a query [33].

While simple techniques such as Naïve Bayes and BM25 have proven successful, more complex machine-learning techniques have also been used to tackle this problem. Guo et al. used a weakly supervised Latent Dirichlet Allocation (LDA) method to mine entities and their associated classes from query log data [17]. At Expedia, Cowan et al. used a conditional random field sequence model to identify locations, names, and amenities in queries for the travel domain [11].

Each of the previous entity recognition techniques rely on the relatively limited vocabulary size for languages such as English. For alphanumeric searches (e.g. for part numbers, employee IDs, telephone numbers, dates, etc.), probabilistic models are less effective as the larger number of possible tokens makes it more likely that the search term has not been seen before. For this situation, some form of pattern matching is often used. Expedia uses regular expressions and heuristic rules for identifying dates, times, and other classes of travel-related concepts [11]. Additionally, both Chang and Manning [7] and Li et al. [26] have reported success using regular expression in the information extraction task. Regular expressions (in addition to simple trigger words) are also used to trigger instant answers in the Duck Duck Go search engine [44].

2.7 Query Classification

In addition to assigning meaning to query segments, it can also be useful to assign meaning to a query as a whole. This process is known as query classification and generally involves assigning a query to one of a finite number of classifications representing the topic or intent of the query. For example, Wan at Target noticed that queries such as *7 ring check binder* were returning pieces of jewelry instead of just office supplies [45]. Target addressed this problem by training a Naïve Bayes classifier (such as the one described in Section 2.2) using manually labeled queries from the query log in order to map queries to Target's two-level product classification scheme. Similarly, Wayfair also improved search relevancy using a Naïve Bayes classifier [8]. As always, other approaches have been considered. Le and Bernardi achieved success training a SVM classifier using clickstream data [25]. Kouylekov et al. leveraged vector space ranking models (similar to those discussed in Section 2.1) to compare the documents returned by a query to a canonical category document derived from Wikipedia data [23]. Finally, Beitzel et al. explored using neural networks and selection preference strength algorithms for classification [4].

2.8 Collection Enrichment

With each of these query expansion techniques, there is a risk that the vocabulary used in the collection does not match that of the individual searching for information. As a result, query expansion techniques (especially ones like pseudo relevance feedback discussed in Section 2.3) might produce the wrong expansion terms and hurt performance. In these situations, it can be useful to leverage external data sources to generate the language model for query expansion. However, as was noted earlier with WordNet, external content sources can also hurt performance when they use a word in a different context or with a different meaning than is used within the document collection. Peng et al. researched this problem in two different papers and concluded that collection enrichment can be successful

in the enterprise domain when targeted at specific queries that would most benefit from it [30, 31]. In particular, they only used external collections (such as Wikipedia) when standard query performance predictors indicated that these collections would improve performance. Similarly, He and Ounis explored selectively switching between local and external query expansion [19]. There an Average Inverse Collection Term Frequency metric was used to determine which collection was more useful for expansion or whether expansion would improve performance at all.

3 Methodology

3.1 Problem Queries

To focus query expansion efforts, the existing query logs at the manufacturing company were mined to identify classes of queries that should have higher relevancy to improve user satisfaction. Several such classes were identified. They are represented by the following queries:

- *151-99* should be recognized as a part number and alternative forms should be included in the suggestions. (Discussed in Section 3.2)
- *mike james street* should be parsed as *first_name = mike* and *location = james street*. (Discussed in Section 3.3)
- *vacation request* should be interpreted as a query for documents from the Human Resources department. (Discussed in Section 3.4)
- *transfer* is a vague query. It likely represents a shortened form of the query *transfer request* which should also target documents from the Human Resources department. (Discussed in Section 3.5)
- *summer picnic* should match documents about the *company picnic* (Discussed in Section 3.6)

It is worth considering each of these query forms in more depth.

3.2 Alphanumeric Searches

As noted in Section 2.6, regular expressions are commonly used in information retrieval, particularly for identifying entities with a limited set of known patterns (e.g. dates and

times). At the manufacturing company being studied, there are over 60 different alphanumeric codes present in the various databases. A subset of these are shown in Table 2. Uniquely identifying each code type would likely improve the relevancy of search results. While Expedia previously maintained heuristic rules including regular expressions for 14 travel-related concepts [11], maintaining such rules for well over 60 concepts would be cost-prohibitive for a small team at the manufacturing company. In addition, some of the patterns are ambiguous. For example, the pattern of 000-0000 (three digits followed by a dash and four more digits) is common to both part numbers and phone numbers. However, some of the ambiguities can be resolved by knowing probable values for the various segments. For example, the number 999-1234 is likely to be a part number, while the number 555-1234 is likely to be a phone number.

Table 2: Sample alphanumeric codes and their variants.

Code Type	**Example Variants**
Part Number: Company 1	999-0123-000, 999-0123, 999-123
Part Number: Company 2	99990 - 01234-0, 99990-01234-0, 99990012340
Phone Number	+16105551234, (610) 555-1234, 610-555-1234, 555-1234
ECO Number	E0001234, E*1234
Manufacturing Line	FA001
VIN	1C3CCBCG0CN999999

Neural networks present another approach that has shown success in pattern recognition tasks. Deep neural networks have gained significant traction in the information retrieval field where they have been leveraged for complex analysis of linguistics and categorization of images segments. Yin et al. even used neural networks to derive SQL queries from natural language queries [50]. Neural networks have also been used in query classification [4]. Based on the success of neural networks within related fields, a neural network was constructed for classifying the various alphanumeric codes present at the manufacturing company. Once a given term was correctly classified, different heuristics were used to expand the alphanumeric code to different forms and/or to boost results of a particular type.

3.3 Entity Recognition

The Naïve Bayes classifier proposed by Kim et al. [22] provides an attractive approach for identifying the probable fields searches present in each query. However, deploying it would require developing a mechanism for storing the learned model in memory and/or on disk. Apache Solr and Lucene already provide a fast and scalable search index which features in-memory caching and disk storage structures. They also implement the BM25 ranking algorithm which has been successful in the related named entity recognition task and has been shown to be very similar to Naïve Bayes as discussed in Section 2.2. Therefore, a secondary inverted index was developed which indexes each field-value pair from the primary index along with the frequency of that pairing. After applying a boost on the frequency field while searching, the final ranking algorithm becomes

$$\text{score}(q, d) \approx \log_{10}(\text{freq}(d)) + \sum_{t \in q} \text{idf}(t, d) \tag{7}$$

(Note that the length normalization and term frequency terms of the BM25 algorithm are omitted for simplicity as the field values average only a couple of words long.) Using Apache Solr also offered additional benefits. The built-in analysis tool chains made it easy to implement Porter stemming, stop-word filtering, and more. In addition, the highlighting module helped when matching results with the terms in the original query.

3.4 Intent Classification

While expanding queries using identified entities and alphanumeric code types improved the performance of some queries, it also harmed the performance of other queries. For the *vacation request* query, the entity recognition expansion module included misleading matches, such as for the document type *Supplier Request*. Since query classification has

been successful in boosting search results that match the class of the query (as noted in Section 2.7), the same technique can be used to boost query expansion suggestions that match the class of the query. Therefore, a module was added to the pipeline that compares the distribution of classes for the query as a whole to the distribution of classes for each considered metadata expansion. If the distributions of the expansion and the query as a whole were significantly different, the suggestion was discounted to keep it from harming the performance of the query.

3.5 Collection Enrichment

Ambiguous queries pose another problem for query expansion (and relevancy in general). As was noted in Section 2.8, using external document collections as a source of query expansion terms can be useful in these scenarios. However, this approach must be used with care as there is a significant probability that differences in the language models between the primary collection and the external collection will result in decreased performance for some queries. Therefore, initial research into leveraging collection enrichment focused on how it can help with modifying expansion suggestions produced by other modules instead of with generating suggestions. In particular, if the query classification module could not confidently predict the classification of the query, collection enrichment was used to provide additional terms to help with the classification process. While research (such as that performed by Peng et al. [30]) has shown that Wikipedia can be a useful external collection for enterprise search applications, the data available as part of Microsoft's Concept Graph[9] provided a compelling option as much of the work in processing the raw data and identifying topics had already been performed [47, 48]. Therefore, this data set was used in the initial research.

[9] `https://concept.research.microsoft.com`

3.6 Word Vectors

While a manual thesaurus does help with some query expansion, it does not cover all possible mismatches between the vocabulary used by searchers and document authors. As a result, techniques for automatically identifying similar words to those in the query were also considered. Shalaby et. al. [37] and Diaz et al. [14] each were able to improve relevancy using search systems which leveraged Google's word2vec. This library aims to learn a vector representation for each word in the corpus such that words used in similar contexts (and presumably with similar meanings) have parallel vectors with a cosine similarity near to one. For this investigation, the fastText library from Facebook was used to generate word vectors. While similar to word2vec, this library differentiates itself by acting on word substrings instead of entire words. It was chosen due to its performance (as measured by Bojanowski et al. [5]) as well as the availability of Windows builds for the library. With some small tweaks, these Windows builds could be modified for easy interoperability with C# code[10]. Targeted collection enrichment was also considered to improve the quality of the learned word associations.

3.7 Architecture and Experimental Design

As was mentioned in Section 1.3, the focus of the study is a custom Apache Solr-based enterprise search application deployed at a local medium-sized manufacturing company. This application indexes over a dozen databases and document repositories at the company. It features a custom user interface written in C# that interacts with Solr using the REST API. The C# interface is responsible for parsing the query (supporting a subset of Lucene's query syntax[11]), checking spelling using a custom spell-checking module, and suggesting synonyms from a manually-constructed thesaurus. After performing these func-

[10] `https://github.com/erdomke/fastText`

[11] `https://lucene.apache.org/core/6_5_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package.description`

tions, the interface attempts to expand the query further by passing the query through the five additional expansion modules proposed in the previous sections. This architecture is shown in Figure 3. Afterwards, the resulting query is submitted to the Solr REST API. The results are then formatted and presented to the user.

To assess the impact of these modules on search relevancy, a set of labeled queries was needed. In particular, a list of relevant documents had to be identified for each of the selected queries. While a small test set was developed manually, additional help was needed. Therefore, search users were enlisted to manually label relevant results for a random subset of their searches. An example of such a survey is shown in Figure 4. This survey was presented randomly on approximately 1 out of every 15 searches. It asked the user to indicate for each of the top 5 ranking documents if the document was useful to the user's search need. This data was added to the list of manually selected and labeled search queries for use with the search engine analysis.

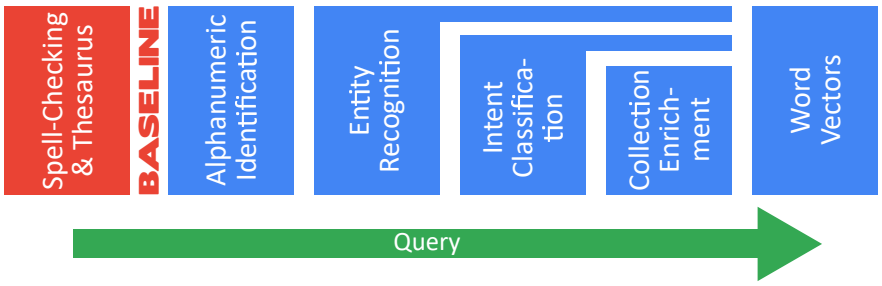

Figure 3: Architectural diagram of the query expansion pipeline.

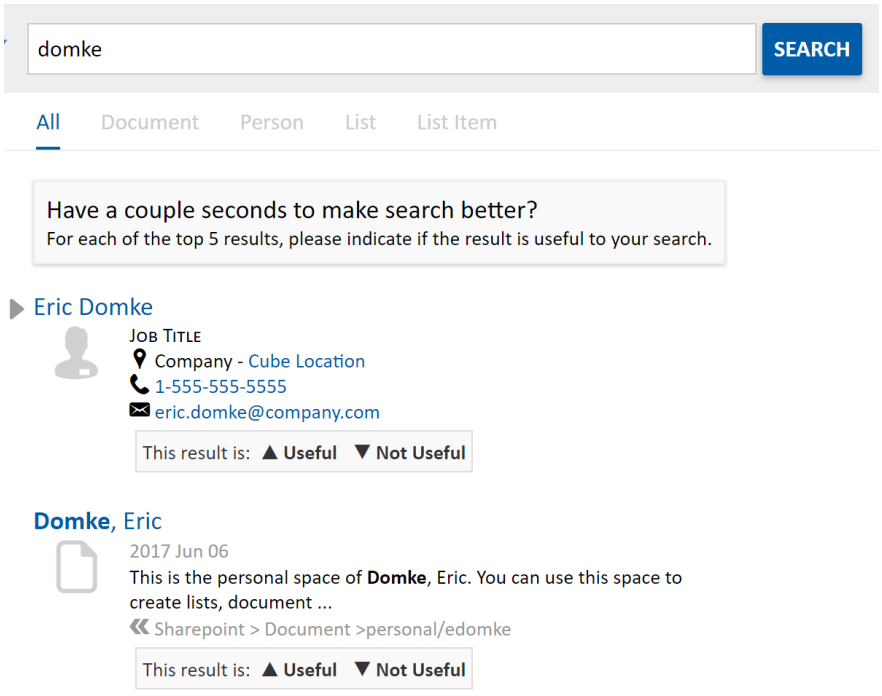

Figure 4: Screenshot of relevance survey

4 Implementation

4.1 Solr Configuration

The Solr index was configured with two primary full-text fields representing the title and body (i.e. description) of each document/record. When a document was added to the Solr index, the contents of these two fields were processed such that

- Accented characters were replaced with their root Latin characters.
- Tokens were identified by splitting the text on white space, case changes, and transitions between letters and digits.
- English stop words were removed.
- Tokens were stemmed using the Porter stemmer.

A third 'exact match' field was used to store terms and/or codes that uniquely and precisely describe the document. Searches against the Solr index were parsed using the `edismax` parser and were sent to these three fields–`title`, `description`, and `exact_match`.

4.2 Federated Search

To handle permissions appropriately, the search solution also federated searches to other services. In particular, each search was sent to two different SharePoint installations in addition to the Solr index. The advantage to this was that all the content of each SharePoint installation did not need to be indexed regularly. In addition, the SharePoint search already honored permissions, so permissions didn't have to be indexed and calculated separately. The disadvantage is that it became more difficult to determine how to interleave the results returned by the SharePoint servers with each other and the main index, especially if these search engines were poorly tuned. To help fix this, a publicly-accessible

subset of the data available in each SharePoint installation was indexed in the Solr index. This way, the relevancy scores for SharePoint documents returned by the Solr index could be compared to the scores assigned to those same documents by SharePoint, so that the final rank of the SharePoint documents could be determined.

4.3 Spelling and Thesaurus Module

To provide spelling suggestions, a custom spell-checking module was developed leveraging code provided by Faroo[12]. This module identified all the suggestions within an edit distance of two and ranked them using a probability model. This ranking model included

- the minimum probability of each edit required to transform the typed word into the candidate correction (e.g. the probability of substituting an e for an i).
- the probability that the correct word would have a Damerau-Levenshtein edit distance of d from the typed word.
- the probability that the correct word would have a double metaphone phonetic key of K_c given the phonetic key of the typed word K_t. In particular, if the phonetic keys were identical, return that probability. If the phonetic keys were different, return the probability that the correct word would have a key starting with the letter L_c given that the typed word's phonetic key started with the letter L_t.

The word with the highest probability was ranked first in the list of spelling suggestions. The various probabilities were computed using the misspelling data compiled by Peter Norvig[13]. Word splits and combinations were also considered when attempting to identify the correct words. Beyond spelling, the module also used a manually constructed thesaurus to add synonym suggestions. This was compiled from multiple sources including

[12] `https://github.com/wolfgarbe/symspell`
[13] `http://norvig.com/ngrams/spell-errors.txt`

internal acronym lists, the Defense Technical Information Center Thesaurus[14], the Transportation Research Thesaurus[15], and more.

4.4 Alphanumeric Identification Module

To identify alphanumeric codes, a neural network was constructed. The best success was observed using a shallow neural network with 116 input neurons, 61 hidden neurons and 61 output neurons. The first 57 input neurons represented the first 19 characters of the input. Each character used three neurons. The first contained a value scaled between either -1.0 and -0.2 or 0.2 and 1.0 if the character was a letter or 0 otherwise. The second contained a similarly scaled value if the character was a digit. The third contained a similarly scaled value if the character was one of 32 common ASCII symbols. The next 57 input neurons represented the last 19 characters of the input, again using three neurons per character. Having each number represented both left-aligned and right-aligned was useful because some alphanumeric codes are truncated by dropping characters at the end (e.g. part numbers) while other are truncated by dropping characters at the beginning (e.g. area codes for phone numbers). Finally, the last two neurons represented the total length of the string and the number of alphanumeric characters in the string respectively. The output neurons were set such that the neuron representing the category of the input had a value of 1.0 while all other neurons had a value of -1.0.

Training proceeded using 19,654 training examples. After every 50 epochs of training, the accuracy was measured on a validation set of 17,356 examples, and the state of the neural network was saved. The saved network that had the highest accuracy on the validation set was preserved for further use. The most accurate network to-date exhibited an accuracy of just over 85% on the validation set.

The classification of the alphanumeric codes was not used directly to expand the queries. Rather, hard-coded rules dictated how the classification was used. Some rules were used

[14] `http://www.dtic.mil/dtic/services/dtic_thesaurus/download.html`
[15] `http://trt.trb.org/trt_download.asp`

to boost documents of a given type. For example, if the code was identified as a part number, documents of type `Part` were given a boost. Other rules normalized the format of a particular code before suggesting a particular field/value search. For example, a search for `+16105551234` identified as a phone number was converted to a search for `cell:610-555-1234`.

4.5 Entity Recognition

Each search was sent to a separate Apache Solr index where each 'document' in the index consisted of a field name, value, frequency, and cluster distribution (see Section 4.6). Solr highlighting was used to help align the returned documents with the query. This matching process started with the phrases consisting of the most words and progressing from there. In the process, multi-word tokens were merged into phrase searches where appropriate. Suggestions were ranked based on their BM25 relevancy scores and the number of documents having the particular field value.

In addition to merely suggesting metadata expansions for the query, the following specific rules were included:

- If a one or two letter term followed a word identified as a `first_name`, a prefix search on the `last_name` field was added as a suggestion for the one/two letter term.
- The scores of secondary matches $\{m_2 \dots m_n\}$ for a given field were suppressed to promote diversity of suggestions instead of having multiple suggestions from the same field. Consider the `mike james street` query. `James street` has high relevancy matches for `location:james street` and `department:james st` along with lower relevancy matches for `location:main street` and `location:washington street`. In order to suppress the lower relevancy matches, the scores of these secondary matches are suppressed.
- Consider the following JSON documents in the index:

```
{ title: "Mike Smith",
  type: "Person",
  first_name: "Mike",
  location: "James Street" }
{ title: "James Street",
  type: "Building" }
```

The query `mike james street` should produce a high score for the first document. When it is parsed to `first_name:Mike` and `location:James Street` (as described earlier), the relevancy is improved. However, the query `james street` should produce a high score for the second document. If it used the same parsing rules, it would be parsed to `location:James Street`, which would also favor the first document, thereby decreasing the relevancy of the results. To counteract this, a rule was put in place such that when a parsed query yielded a single phrase (`james street`), the field matches (`location`) were not used directly. Rather, the fields which were matched (e.g. `location`) were used as to identify which types of documents to boost (e.g. `type:Building`).

4.6 Intent Classification Module

To suppress unrelated field-mapping suggestions, a module was built that aimed to classify queries into one of several clusters. To do this, each document was assigned a cluster when it was indexed. For documents created from database records, the cluster determination was largely based on the type of the record. For documents created from unstructured data stored in web pages and MS Office documents, the cluster determination was based on the department which generated the data (inferred by the location where the data was stored and other means). In addition, fields were added to the primary index for storing word count statistics for each document. Finally, a cluster field was added to the

secondary field-mapping index described in Section 4.5. This field contained a delimited list of how many documents in each cluster had the particular field value.

When executing a query, term frequency statistics were computed for each query term relative to each cluster within the primary index. Using this data (along with the word count data), an approximate BM25 score (from Equation 4) was computed for each cluster. These scores were then normalized by computing the ratio of each score to the total score of all clusters. By performing this query directly against the primary index, data from the most recently indexed documents was used and another index did not need to be built. However, having another index of meta-documents (where each document consisted of the concatenation of all the documents in each cluster) would potentially make the computation more accurate as it would allow for the use of Solr phrase boosts to help with compound words in the query.

To determine the similarity (or distance) between a query term and a field-mapping suggestion, the cluster distributions of the two were compared. Consider $\boldsymbol{C}$ to be set of clusters to which query term q may belong ordered by BM25 score in descending order. As a result, the distance between a given suggestion s and this cluster set $\boldsymbol{C}$ is given by

$$\text{dist(C,s)} = \sum_{i=1}^{5} \frac{\text{BM25}(c_i)}{\sum_{c \in \boldsymbol{C}} \text{BM25}(c)} - \frac{\text{docCount}(c_i, s)}{\text{docCount}(s)} \tag{8}$$

where $\frac{\text{BM25}(c_i)}{\sum_{c \in \boldsymbol{C}} \text{BM25}(c)}$ is the normalized BM25 score of cluster c_i discussed earlier, $\text{docCount}(c_i, s)$ is the number of documents in cluster c_i containing suggestion s and $\text{docCount}(s)$ is the total number of documents containing suggestion s. If this distance was greater than an empirically defined threshold, the probability of the suggestion was significantly discounted, effectively removing the suggestion from the list.

4.7 Collection Enrichment Module

In some situations, the cluster assignments for a query were not obvious due to the ambiguous nature of the query. In particular, the highest-ranking cluster might have a normalized BM25 score less than 0.1. To deal with queries without an obvious cluster, the query was expanded using concept labels derived from the Microsoft Concept Graph. Another Solr index was built where each document included a word or phrase, its concept label, the frequency of the word/phrase ($valueFreq$), the frequency of the label ($labelFreq$), the frequency of the value with that label ($freq$), and a score. The score was computed using a formula similar to the $Rep(e, c)$ Basic-level Conceptualization score introduced by Wang et. al [47]. The score is given by

$$score = 2 \cdot \ln freq - \ln labelFreq - \ln valueFreq \tag{9}$$

With this setup, the index could be queried for different values using the score field as a boost to the traditional BM25 scoring. The different concept labels returned were then used to expand the original cluster query in an attempt to get a more accurate cluster representation. This new cluster representation was then used by the intent classification module to determine which suggestions to demote.

4.8 Word Vectors

To compute the word vectors, all the indexed documents were concatenated into a single document on disk. This document was then used to train a fastText model of the vocabulary. Due to issues with special characters, many non-ASCII characters were converted into comparable ASCII characters and many punctuation characters were replaced with spaces. In addition, all the text was converted to lower case. Initially, this model did not return related words that seemed sufficiently useful for some of the test queries in domains

such as Human Resources, Travel, and Information Technology. As a result, collection enrichment was considered. Over various iterations, additional content was added from

- corporate policies from other institutions. These policies were obtained by searching Google for documents using queries such as *employee handbook*, *travel policy*, *transfer request*, *reasonable suspicion policy*, and *acceptable use policy*
- the Super User site[16] on Stack Exchange
- select Wikipedia articles (e.g. on 401(k), stock options, etc.)
- internal query logs

In addition, it was observed that vectors should be learned for compound words as well. For example, the concept *human resources* should have a very different vector representation relative to the vectors of the constituent words, *human* and *resources*. To identify the compound words within the corpus, a list of compound words was derived from the manual thesaurus, the labels from Microsoft's Concept Graph, and the terms in WordNet. The final document used for learning the fastText model included two copies of the enriched corpus – one where the spaces between compound words were replaced with underscores (e.g. *human resources* → *human_resources*) and one where compound words were not modified. Training with this document continued for 8 epochs using substrings ranging from 3 to 8 characters and a dictionary size of 35,000 buckets. Limiting the number of buckets helped to cap the size of the model when stored on disk and in memory.

When executing a query, the fastText model was loaded into memory and queried for the words nearest to each query word (cosine similarity nearest to 1). To account for context, similar two word phrases (bigrams) were also considered. As an example, consider a search for *corporate phone directory*. To find the words similar to *phone*, the model was queried for all the words similar to *phone*, all the bigrams similar to *corporate_phone* and

[16] `https://superuser.com/`

starting with *corporate_* and all the bigrams similar to *phone_directory* and ending with *_directory*. For each of the resulting words or phrases W, a cosine distance was computed between the pairs (W, *phone*), (*corporate_*W, *corporate_phone*), and (W*_directory*, *phone_directory*). The resulting words were then ordered by ascending variance. That is, given query Q consisting of $\{q_1, ..., q_m\}$ and the word expansions E_i consisting of $\{e_{i,1}, ..., e_{i,n}\}$ for the word q_i, the variance would be given by

$$\begin{aligned} var(e_{i,j}, Q) = &(1 - \text{cosineDist}(\overrightarrow{q_{i-1}, q_i}, \overrightarrow{q_{i-1}, e_{i,j}}))^2 \\ &+ (1 - \text{cosineDist}(\overrightarrow{q_i}, \overrightarrow{e_{i,j}}))^2 \\ &+ (1 - \text{cosineDist}(\overrightarrow{q_i, q_{i+1}}, \overrightarrow{e_{i,j}, q_{i+1}}))^2 \end{aligned} \tag{10}$$

5 Results

5.1 Relevancy Metric Selection

Several metrics are available to compare the effectiveness of the different query expansion approaches. Since an ideal search engine has both high precision (a high percentage of the returned results are relevant) and high recall (a high percentage of the relevant results are returned), an F score is a commonly used metric. The F score balances precision, P, and recall, R, according to the formula

$$F_\beta = \frac{(1+\beta^2)PR}{\beta^2 P + R} \tag{11}$$

where β indicates the balance between precision and recall. For this analysis, an F_2 score (where $\beta = 2$) might be appropriate as it places more weight on recall over precision. However, the score treats the results as an unordered set and does not indicate if the relevant documents are at the top of the results or the bottom. An alternative metric is the normalized discounted cumulative gain, $nDCG$. This metric starts with the actual discounted cumulative gain at position p computed using the formula

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{12}$$

where rel_i is the relevancy score for the result at position i. The metric then compares this value to the ideal discounted cumulative gain which is computed using the same formula but assuming an ideal result set consisting only of relevant results ordered descending by relevancy. Therefore, the normalized discounted cumulative gain is given by the formula

$$\mathrm{nDCG_p} = \frac{\mathrm{DCG_p}}{\mathrm{IDCG_p}} \tag{13}$$

For this investigation, the $\boldsymbol{nDCG_{10}}$ score was used to assess the quality of the top 10 documents returned by the search engine.

5.2 Alphanumeric Identification

To assess the impact of each module on relevance, different configurations of the search application were tested. For each module, a test was performed where only that module and its dependent modules (including the spell-checking and thesaurus module) were enabled. The resulting relevance scores across the 182 test queries (derived from manual labeling and the explicit relevance feedback, see Section 3.7) were then compared to the scores obtained when only the dependent modules were enabled. The first module to be tested in this manner was the alphanumeric identification module. This module showed an improvement in its $\boldsymbol{nDCG_{10}}$ relevance over the spell-checking baseline of 0.046, which is statistically significant with over 95% confidence. A breakdown of the module's performance is given in Figure 5. This graph shows a box plot of the $\boldsymbol{nDCG_{10}}$ scores relative to the spelling module for each query category. From this, it can be seen that the alphanumeric identification module helped with both person searches and part number searches, but the module hurt relevancy for many entity searches. This effect can be seen in more detail by examining Figure 6. This graph traces the cumulative impact of each successive query expansion module on the $\boldsymbol{nDCG_{10}}$ scores for each alphanumeric query in the test set. The blue person lines labeled 'A' in Figure 6 demonstrate the beneficial impact this module had on person searches, particularly on searches for people by employee ID. The negative impact on entity searches can be seen in the green lines labeled 'B' in this same figure. An example of such a search is `issue 12345` where `issue` refers to the database record

of a quality issue. In most cases, the neural network correctly identified the code `I:12345` as pertaining to a quality issue. However, without the prefix, the network assumed that the number `12345` in isolation referred to an employee by ID. Therefore, the module erroneously boosted results for people in general and the result for employee 12345 more specifically, thereby hurting relevance. As the graph shows, the entity recognition module helped to compensate for this error by identifying the word `issue` as a type of document.

5.3 Entity Recognition

The entity recognition module improved the $nDCG_{10}$ relevance scores over the spell-checking baseline by 0.051, which is also statistically significant at 95% confidence. Box plots of the module's performance for each query category relative to the spell-checking module baseline are given in Figure 7. These plots show that the module performed well with both person and entity searches. In particular, it could correctly identify first names and last names in person queries as well as document type names in entity queries. As a result, relevant documents were boosted with the query expansion. However, many corporate searches performed worse when this module was used. An example of this is the query `vacation request`. In this situation, the module inappropriately identified `type:"Supplier Request"` as a valid expansion candidate for the word `request`. Since these types of requests are very different, irrelevant results were boosted in the result set.

5.4 Intent Classification

The impact of the intent classification module was measured relative to the entity recognition module since it only modified the output of this module. From this, it can be seen that the intent classification module improved search relevancy with a 0.020 increase in the average $nDCG_{10}$ score relative to the entity recognition baseline. While this is a noticeable improvement, the module only had an impact on 5 queries. As a result, it is difficult to assess the significance of this change. A paired t-test indicates the result was significant

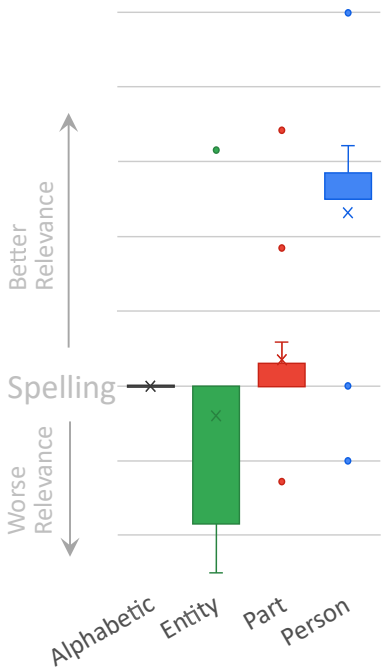

Figure 5: Average $nDCG_{10}$ difference between using the alphanumeric identification module and only using spell checking for different search categories. All non-alphanumeric searches are categorized as "alphabetic".

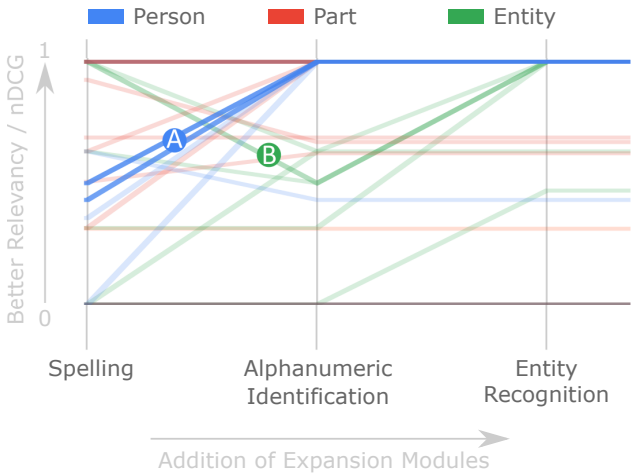

Figure 6: Traces of $nDCG_{10}$ scores for each alphanumeric query after the addition of each query expansion module.

at 90% confidence, but not at 95% confidence. Traces of the $nDCG_{10}$ scores for the queries impacted by this module are shown in Figure 8. These traces show that the module only improved the relevancy of queries and that most of these improved queries were corporate queries, such as the `vacation request` query mentioned earlier. For this query, the module could correctly identify that the `type:"Supplier Request"` candidate expansion was sufficiently different from the intent of the query and hence suppressed it.

5.5 Collection Enrichment

Qualitatively, Microsoft's concept graph appeared useful at identifying suggestions for ambiguous queries. This can be seen with the suggested concept labels for the query `transfer` shown in Table 3. These labels appear to correctly identify the query as being related to Human Resources. However, relevancy measurements for the collection enrichment module (which further modified the intent classification module) showed that the module only impacted one query in the test set, and generally had no discernible impact on relevancy relative to the intent classification module.

Table 3: Microsoft concept graph labels for "transfer".

Concept Label	**Score**
personnel action	0.444
routine personnel action	0.080
record employee information	0.076
call handling button	0.075
selection decision	0.060
non sale conveyance	0.058

5.6 Word Vectors

Before testing relevancy of the word vector module, several different models were developed and tested against different search terms to determine the model with the best qualitative performance. The models were trained using only data from Wikipedia[17], only data

[17] `https://fasttext.cc/docs/en/pretrained-vectors.html`

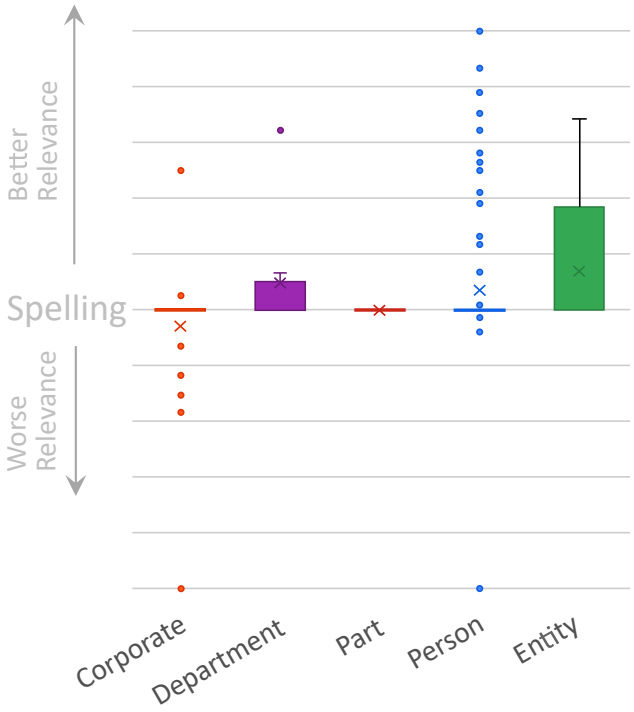

Figure 7: Average $nDCG_{10}$ difference between using the entity recognition module and only using spell checking for different search categories.

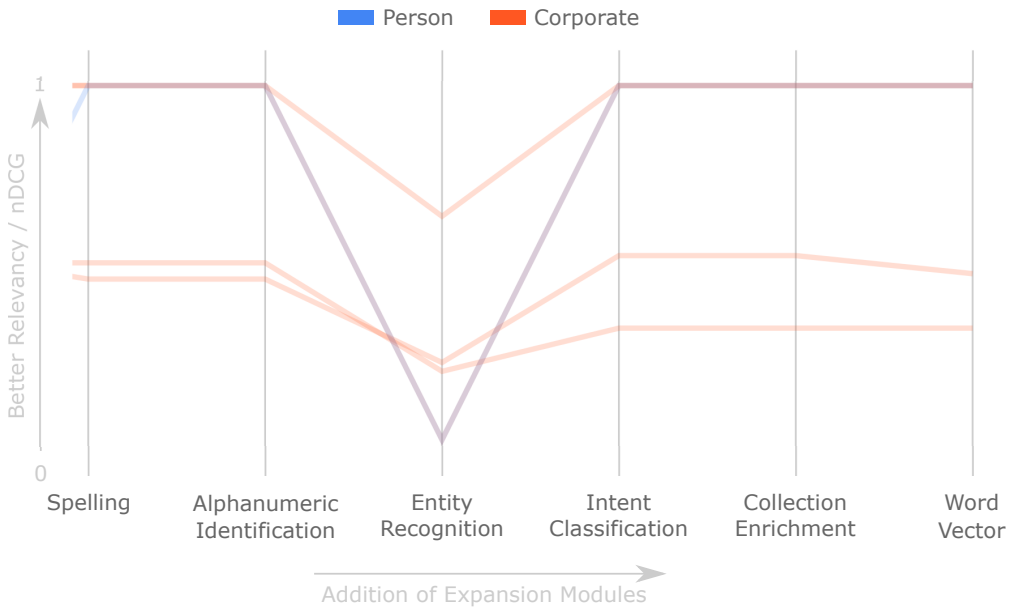

Figure 8: Traces of $nDCG_{10}$ scores for query impacted by the intent classification module.

from the document collection, data from the document collection with selective external enrichment (e.g. select Wikipedia articles, employee handbooks from other companies, etc.), and the enriched document collection with compound words combined into a single token. Examples of a nearest neighbor search using these models is shown in Table 4. As can be seen by the nearest neighbors to `mike`, the Wikipedia model produced words which are not closely related (like `doug`) as well as words with superfluous characters. A more useful suggestion would be the name "michael" for which "mike" is a common nickname. This suggestion showed the closest affinity for the search word in the final model (which utilized collection enrichment and compound word identification). While this final model appears useful, relevancy testing showed that even with this model, the word vector module caused the average $nDCG_{10}$ relevancy to decline by 0.019 relative to the spell-checking baseline. This decrease in relevancy occurred across all categories of searches as the box plots in Figure 9 demonstrate. This was likely because while the word vector module did indeed suggest related words, these words were not present in the relevant documents. For example, given a search for `john smith`, the word vector module might suggest `jane doe`. While Jane does indeed work with John, the employee record for "John Smith" (which is the most relevant document) contains no mention of "Jane Doe".

Table 4: Word vector expansion suggestions for "tuition" and "mike". Last names have been omitted for privacy.

Wikipedia	**Collection**	**Collection+**	**Collection+ Compound**
`tuition`			
tuitions	payment	reimbursement	tuition_reimbursement
tuitioned	reimbursement	reimbursements	reimbursement
tuiti	reimbursements	payment	reimbursements
tuition/fees	payments	reimburses	payments
scholarships	adoption	enroll	medical_reimbursement
`mike`			
doug	{*last name*}	{*last name*}	mike_{*last name*}
Çámike	{*last name*}	{*last name*}	mike_{*last name*}
dave	{*last name*}	{*last name*}	michael
pete	{*last name*}	michael	{*last name*}
brando/mike	{*last name*}	{*last name*}	michael_{*last name*}

5.7 Overall Assessment

As shown in Figure 10, the overall impact of all five modules was positive with the average $nDCG_{10}$ score increasing from 0.65 to 0.77. A two-tailed paired t-test indicates that this difference was statistically significant with at least 95% confidence given the p value of 0.000002. Overall, the query modules both increased the number of queries with perfect relevance and decreased the number of queries which did not return any relevant documents within the first 10 results. Projecting these changes onto the distribution of query types currently seen by the search engine (as discussed in Section 1.2) indicates a predicted change in the $nDCG_{10}$ scores as seen by the user from 0.83 to 0.89.

A breakdown of these results by module is shown in Figure 11. This graph shows the cumulative impact of enabling each of the query expansion modules successively. From the graph, one can see that the alphanumeric identification, entity recognition, and intent classification modules all had a measurable positive impact on relevance. However, the collection enrichment module did not impact the results at all, and the word vector module hurt relevancy.

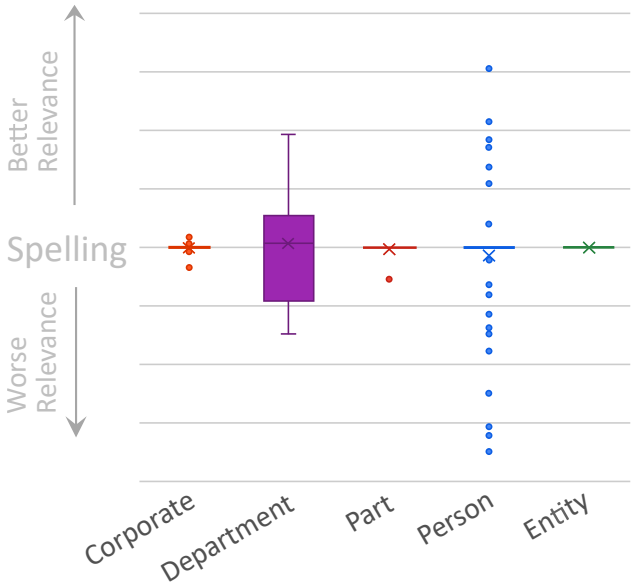

Figure 9: Average $nDCG_{10}$ difference between using the word vector module and only using spell checking for different search categories.

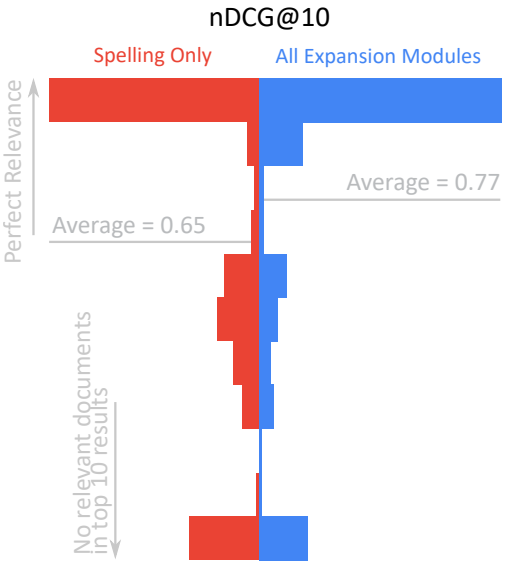

Figure 10: $nDCG_{10}$ histogram showing $nDCG_{10}$ relevancy distribution for only the spelling module (left) vs. all expansion modules (right)

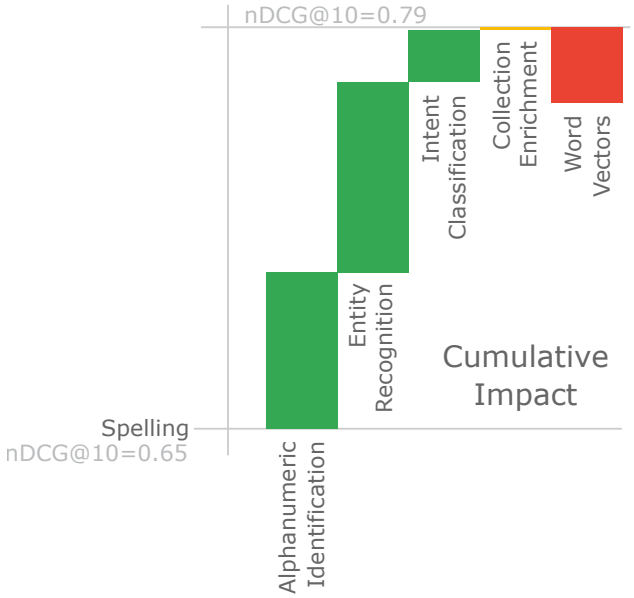

Figure 11: Cumulative $nDCG_{10}$ impact of various query expansion modules

6 Conclusion

From the results, it can be seen that specialized query expansion modules do indeed improve relevancy in enterprise search. In particular, the alphanumeric identification, entity recognition, and intent classification modules all resulted in significant improvements across the test set of queries. These modules are projected to make modest improvements in the relevancy experienced by actual users. These improvements will prove useful as the enterprise search engine being studied currently responds to over 2500 queries from over 120 users every day. These successes allow us to accept the hypothesis that query expansion techniques targeted at enterprise queries are effective in improving relevancy.

Relevancy improvements could not be realized with the collection enrichment and word vector modules. These modules suffered from too narrow of a focus and noise in the expansion suggestions which ended up hurting relevancy more than helping it. Despite these failures, the models did show some initial promise and are worth further investigation. In particular, additional research could help identify techniques for targeting these modules and approaches toward just the queries that would benefit from them. Once properly targeted, these modules could help improve overall relevancy.

By focusing this study on a particular enterprise search application, it was possible to gain deep insights into how the search application was used and what approaches would be most effective at improving relevancy for these uses. In addition, the useful query expansion modules have been deployed for production use, allowing users to benefit from the study. The downside of this focus is that it is uncertain how well the techniques outlined here will generalize to other organizations. An important next step would be to apply these approaches in other enterprise search applications and study their impact on relevancy.

References

[1] Agissilaos Andreou. "Ontologies and Query expansion". PhD thesis. Univ. of Edinburgh, 2005. URL: https://www.icsa.inf.ed.ac.uk/publications/thesis/online/IM050335.pdf (visited on 02/22/2017).

[2] Stephen E Arnold. *Beyond Search Evolution Underway*. Beyond Search. Mar. 1, 2017. URL: http://arnoldit.com/wordpress/2017/03/01/beyond-search-evolution-underway/ (visited on 03/26/2017).

[3] Bissan Audeh. "Experiments on two Query Expansion Approaches for a Proximity-based Information Retrieval Model." In: *CORIA*. 2012, pp. 407–412. URL: https://pdfs.semanticscholar.org/c2fd/b9f5eb7f730137e505de9af7e0aff122e037.pdf (visited on 01/24/2017).

[4] Steven M. Beitzel et al. "Automatic web query classification using labeled and unlabeled training data". In: *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2005, pp. 581–582. URL: http://dl.acm.org/citation.cfm?id=1076138 (visited on 02/25/2017).

[5] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *arXiv:1607.04606 [cs]* (July 15, 2016). arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606 (visited on 08/30/2017).

[6] Wladmir Cardoso Brando. "Exploiting Entities for Query Expansion". PhD thesis. Belo Horizonte, Brazil: Universidade Federal de Minas Gerai, Oct. 2013. 93 pp. URL: http://www.bibliotecadigital.ufmg.br/dspace/bitstream/handle/1843/ESBF-9GMJW2/wladmircardosobrandao.pdf?sequence=1 (visited on 01/17/2017).

[7] Angel X. Chang and Christopher D. Manning. "SUTime: A library for recognizing and normalizing time expressions." In: *LREC*. 2012, pp. 3735–3740. URL: http://www-nlp.stanford.edu/pubs/lrec2012-sutime.pdf (visited on 11/20/2016).

[8] Ben Clark. *Better Lucene/Solr searches with a boost from an external naive Bayes classifier | Wayfair Engineering*. Wayfair Engineering. Oct. 23, 2012. URL: http://engineering.wayfair.com/2012/10/better-lucenesolr-searches-with-a-boost-from-an-external-naive-bayes-classifier/ (visited on 11/16/2016).

[9] Marco Cornolti et al. "The SMAPH system for query entity recognition and disambiguation". In: *ERD '14: Proceedings of the first international workshop on Entity recognition & disambiguation*. ACM Press, 2014, pp. 25–30. ISBN: 978-1-4503-3023-7. DOI: 10.1145/2633211.2634348. URL: http://dl.acm.org/citation.cfm?doid=2633211.2634348 (visited on 11/08/2016).

[10] Tina Costanza. *Global enterprise search market to reach US$4.68bn by 2019 - Frost & Sullivan*. Silicon Republic. Jan. 25, 2013. URL: https://www.siliconrepublic.com/enterprise/global-enterprise-search-market-to-reach-us4-68bn-by-2019-frost-sullivan (visited on 03/26/2017).

[11] Brooke Cowan et al. "Named Entity Recognition in Travel-Related Search Queries." In: *AAAI*. 2015, pp. 3935–3941. URL: https://pdfs.semanticscholar.org/2da4/0f5dda818aea7cca17affa976735c0452cb6.pdf (visited on 01/28/2017).

[12] Hang Cui et al. "Probabilistic query expansion using query logs". In: *Proceedings of the 11th international conference on World Wide Web*. ACM, 2002, pp. 325–332. URL: http://dl.acm.org/citation.cfm?id=511489 (visited on 02/14/2017).

[13] Barb Darrow. *Google Search Appliance: So Long*. Fortune.com. Feb. 4, 2016. URL: http://fortune.com/2016/02/04/google-ends-search-appliance/ (visited on 03/27/2017).

[14] Fernando Diaz, Bhaskar Mitra, and Nick Craswell. "Query expansion with locally-trained word embeddings". In: *arXiv preprint arXiv:1605.07891* (2016). URL: https://arxiv.org/abs/1605.07891 (visited on 01/28/2017).

[15] *Enterprise Track*. Text REtrieval Conference (TREC). Aug. 4, 2016. URL: http://trec.nist.gov/data/enterprise.html (visited on 03/26/2017).

[16] Zhiguo Gong, Maybin Muyeba, and Jingzhi Guo. "Business information query expansion through semantic network". In: *Enterprise Information Systems* 4.1 (Feb. 2010), pp. 1–22. ISSN: 1751-7575, 1751-7583. DOI: 10.1080/17517570903502856. URL: http://www.tandfonline.com/doi/abs/10.1080/17517570903502856 (visited on 01/28/2017).

[17] Jiafeng Guo et al. "Named entity recognition in query". In: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2009, pp. 267–274. URL: http://dl.acm.org/citation.cfm?id=1571989 (visited on 01/28/2017).

[18] Gyeong June Hahm et al. "A personalized query expansion approach for engineering document retrieval". In: *Advanced Engineering Informatics* 28.4 (Oct. 2014), pp. 344–359. ISSN: 1474-0346. DOI: 10.1016/j.aei.2014.04.002. URL: https://www.sciencedirect.com/science/article/pii/S1474034614000317 (visited on 01/28/2017).

[19] Ben He and Iadh Ounis. "Combining fields for query expansion and adaptive query expansion". In: *Information Processing & Management* 43.5 (Sept. 2007), pp. 1294–1307. ISSN: 03064573. DOI: 10.1016/j.ipm.2006.11.002. URL: http://linkinghub.elsevier.com/retrieve/pii/S0306457306001956 (visited on 04/15/2017).

[20] Jeanette Jones. *Various Survey Statistics: Workers Spend Too Much Time Searching for Information*. Cottrill Research. Nov. 8, 2013. URL: http://www.cottrillresearch.com/various-survey-statistics-workers-spend-too-much-time-searching-for-information/ (visited on 04/04/2017).

[21] Jin Young Kim and W. Bruce Croft. "A field relevance model for structured document retrieval". In: *European Conference on Information Retrieval*. Springer, 2012, pp. 97–108. URL: http://link.springer.com/chapter/10.1007/978-3-642-28997-2_9 (visited on 11/08/2016).

[22] Jinyoung Kim, Xiaobing Xue, and W. Bruce Croft. "A probabilistic retrieval model for semistructured data". In: *Advances in Information Retrieval*. Springer, 2009, pp. 228–239. URL: http://link.springer.com/chapter/10.1007/978-3-642-00958-7_22 (visited on 06/03/2016).

[23] Milen Kouylekov et al. "Wikipedia-based Unsupervised Query Classification." In: *IIR*. Citeseer, 2013, pp. 116–119. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.417.2061&rep=rep1&type=pdf#page=123 (visited on 02/25/2017).

[24] Michal Laclavik et al. "A search based approach to entity recognition: magnetic and IISAS team at ERD challenge". In: *ERD '14: Proceedings of the first international workshop on Entity recognition & disambiguation*. ACM Press, 2014, pp. 63–68. ISBN: 978-1-4503-3023-7. DOI: 10.1145/2633211.2634352. URL: http://dl.acm.org/citation.cfm?doid=2633211.2634352 (visited on 11/11/2016).

[25] Dieu-Thu Le and Raffaella Bernardi. "Query classification using topic models and support vector machine". In: *Proceedings of ACL 2012 Student Research Workshop*. Association for Computational Linguistics, 2012, pp. 19–24. URL: http://dl.acm.org/citation.cfm?id=2390335 (visited on 02/25/2017).

[26] Yunyao Li et al. "Regular expression learning for information extraction". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2008, pp. 21–30. URL: http://dl.acm.org/citation.cfm?id=1613719 (visited on 11/20/2016).

[27] Xitong Liu et al. "Exploiting entity relationship for query expansion in enterprise search". In: *Information Retrieval* 17.3 (June 2014), pp. 265–294. ISSN: 1386-4564, 1573-7659. DOI: 10.1007/s10791-013-9237-0. URL: http://link.springer.com/10.1007/s10791-013-9237-0 (visited on 01/15/2017).

[28] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2009. ISBN: 0-521-86571-9. URL: http://nlp.stanford.edu/IR-book/ (visited on 01/14/2017).

[29] Jessie Ooi et al. "A survey of query expansion, query suggestion and query refinement techniques". In: *Software Engineering and Computer Systems (ICSECS), 2015 4th International Conference on*. IEEE, 2015, pp. 112–117. URL: http://ieeexplore.ieee.org/abstract/document/7333094/ (visited on 01/28/2017).

[30] Jie Peng, Ben He, and Iadh Ounis. "Predicting the usefulness of collection enrichment for enterprise search". In: *Conference on the Theory of Information Retrieval*. Springer, 2009, pp. 366–370. URL: http://link.springer.com/chapter/10.1007/978-3-642-04417-5_41 (visited on 01/15/2017).

[31] Jie Peng et al. "A study of selective collection enrichment for enterprise search". In: *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 2009, pp. 1999–2002. URL: http://dl.acm.org/citation.cfm?id=1646286 (visited on 04/15/2017).

[32] Joaqun Prez-Iglesias et al. “Integrating the probabilistic models BM25/BM25F into Lucene”. In: *CoRR* abs/0911.5046 (2009). URL: http://arxiv.org/abs/0911.5046 (visited on 07/14/2016).

[33] Stefan Rd et al. “Piggyback: Using search engines for robust cross-domain named entity recognition”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 965–975. URL: http://dl.acm.org/citation.cfm?id=2002594 (visited on 01/28/2017).

[34] Jason D. Rennie et al. “Tackling the poor assumptions of naive bayes text classifiers”. In: *ICML*. Vol. 3. Washington DC, 2003, pp. 616–623. URL: http://www.aaai.org/Papers/ICML/2003/ICML03-081.pdf (visited on 03/26/2017).

[35] Stephen E. Robertson et al. “Okapi at Trec-3”. In: Text REtrieval Conference. Gaithersburg, USA, Nov. 1994. URL: http://trec.nist.gov/pubs/trec3/papers/city.ps.gz (visited on 03/30/2017).

[36] J Rocchio. “Relevance feedback in information retrieval”. In: *In The SMART retrieval system* (1971), pp. 313–323.

[37] Walid Shalaby et al. “Entity Type Recognition using an Ensemble of Distributional Semantic Models to Enhance Query Understanding”. In: *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*. Vol. 1. IEEE, 2016, pp. 631–636. URL: http://ieeexplore.ieee.org/abstract/document/7552082/ (visited on 01/29/2017).

[38] Jagendra Singh and Aditi Sharan. “A novel model of selecting high quality pseudo-relevance feedback documents using classification approach for query expansion”. In: *Computational Intelligence: Theories, Applications and Future Directions (WCI), 2015 IEEE Workshop on*. IEEE, 2015, pp. 1–6. URL: http://ieeexplore.ieee.org/abstract/document/7495539/ (visited on 01/15/2017).

[39] Min Song et al. “Keyphrase extraction-based query expansion in digital libraries”. In: *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*. ACM, 2006, pp. 202–209. URL: http://dl.acm.org/citation.cfm?id=1141800 (visited on 01/15/2017).

[40] Neha Soni and Jaswinder Singh. “A Detailed Study on Query Expansion Techniques in Information Retrieval”. In: *International Journal of Emerging Trends in Engineering and Development* 4.4 (June 2014), pp. 433–444. URL: http://www.rspublication.com/ijeted/2014/july14/43.pdf (visited on 01/29/2017).

[41] K. Sparck Jones, S. Walker, and S. E. Robertson. “A probabilistic model of information retrieval: development and comparative experiments: Part 1”. In: *Information Processing & Management* 36.6 (Nov. 1, 2000), pp. 779–808. ISSN: 0306-4573. DOI: 10.1016/S0306-4573(00)00015-7. URL: http://www.sciencedirect.com/science/article/pii/S0306457300000157 (visited on 12/13/2017).

[42] Wolfgang Tannebaum and Andreas Rauber. "Using query logs of USPTO patent examiners for automatic query expansion in patent searching". In: *Information Retrieval* 17.5 (Oct. 2014), pp. 452–470. ISSN: 1386-4564, 1573-7659. DOI: 10.1007/s10791-014-9238-7. URL: http://link.springer.com/10.1007/s10791-014-9238-7 (visited on 01/14/2017).

[43] *TFIDFSimilarity (Lucene 6.5.0 API)*. Mar. 2, 2017. URL: https://lucene.apache.org/core/6_5_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html (visited on 03/28/2017).

[44] *Triggers*. DuckDuckHack Docs. URL: https://docs.duckduckhack.com/backend-reference/triggers.html (visited on 04/14/2017).

[45] Howard Wan. "Query Classification for Solr". Lucene/Solr Revolution. Boston, MA, Oct. 14, 2016. URL: https://www.youtube.com/watch?v=ek3ftFfhnWE (visited on 11/16/2016).

[46] Zhongyuan Wang and Haixun Wang. "Understanding Short Texts". In: (2016). URL: https://www.microsoft.com/en-us/research/publication/understanding-short-texts/ (visited on 03/14/2017).

[47] Zhongyuan Wang et al. "An Inference Approach to Basic Level of Categorization". In: ACM Press, 2015, pp. 653–662. ISBN: 978-1-4503-3794-6. DOI: 10.1145/2806416.2806533. URL: http://dl.acm.org/citation.cfm?doid=2806416.2806533 (visited on 03/14/2017).

[48] Wentao Wu et al. "Probase: A probabilistic taxonomy for text understanding". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 481–492.

[49] Zheng Ye and Jimmy Xiangji Huang. "A learning to rank approach for quality-aware pseudo-relevance feedback". In: *Journal of the Association for Information Science and Technology* 67.4 (Apr. 2016), pp. 942–959. ISSN: 23301635. DOI: 10.1002/asi.23430. URL: http://doi.wiley.com/10.1002/asi.23430 (visited on 01/21/2017).

[50] Pengcheng Yin et al. "Neural enquirer: Learning to query tables". In: *arXiv preprint arXiv:1512.00965* (2015). URL: https://pdfs.semanticscholar.org/5a82/9f63a9a03be652ed8568ab6ce77ef0f2a712.pdf (visited on 02/08/2017).